A DYNAMIC LOAD BALANCING APPROACH FOR SOLUTION ADAPTIVE FINITE ELEMENT GRAPH APPLICATIONS ON DISTRIBUTED SYSTEMS

Maheen Islam[1] and Upama Kabir[2]
Department of Computer Science and Engineering
[1]East West University and [2]Dhaka University, Dhaka, Bangladesh Dhaka, Bangladesh.
Email: maheen28_i@yahoo.com and upama2@yahoo.com

Abstract: *Load balancing is the key to the efficient operation of distributed systems. To efficiently utilize computing resources provided by distributed systems, an underlying Dynamic load balancing (DLB) scheme must address both heterogeneous and dynamic features of distributed systems. In this paper, a DLB scheme for Solution Adaptive Finite Element Graph Applications on distributed systems is proposed. Experiments show that by using the proposed distributed DLB scheme which considers the heterogeneous and dynamic features of distributed systems, the execution time and the number of process migration is close to using Condensed Binary Tree Load Balancing (CBTLB) scheme which does not consider the heterogeneous and dynamic features of distributed systems.*

Keywords*: dynamic load balancing, distributed systems, heterogeneity, dynamic network loads.*

1 Introduction

The Finite Element Graph (FEG) method is widely used for the structural modeling of physical systems. In the finite element model, an object can be viewed as a finite element graph, which is a connected and undirected graph that consists of a number of finite elements. Each finite element is composed of a number of nodes. Due to the properties of computation-intensiveness and computation-locality, it is very attractive to implement the finite element method on distributed memory multicomputers [4, 5, 14, 19, 20]. In the context of parallelizing a finite element application program that uses iterative techniques to solve system of equations [1], a parallel program may be viewed as a collection of tasks represented by nodes of a finite element graph. Each node represents a particular amount of computation and can be executed independently. To efficiently execute a finite element application program on a distributed memory multicomputer, we need to map nodes of the corresponding finite element graph to processors of a distributed memory multicomputer such that each processor has approximately the same amount of computational load and the communication among processors is minimized. Since this mapping problem is known to be NP-complete [12], many heuristic methods were proposed to find satisfactory suboptimal solutions [2-5, 13].

For a solution-adaptive finite element application program, the number of nodes increases discretely due to the refinement of some finite elements during the execution. This may result in load imbalance of processors. So, execution of FEG applications on distributed systems involves dynamically distributing the workload among the systems at runtime. A distributed system may consist of heterogeneous machines connected with heterogeneous networks; and the networks may be shared. Therefore, to efficiently utilize the computing resources provided by distributed systems, the underlying dynamic load balancing (DLB) scheme must take into consideration the heterogeneous and dynamic features of distributed systems. DLB schemes have been researched extensively, resulting in a number of proposed approaches [7-10, 15, 17, 18].

However, most of these approaches are inadequate for distributed systems. For example, some schemes assume the multiprocessor system to be homogeneous, (e.g. all the processors have the same performance and the underlying networks are dedicated and have the same performance). Some schemes consider the system to be heterogeneous in a limited way (e.g. the processors may have different performance but the networks are dedicated). To address the heterogeneity of processors, a widely-used mechanism is to assign a relative weight which measures the relative performance to each processor. For example, Elsasser [16] generalize existing diffusive schemes for heterogeneous systems. Their scheme considers the heterogeneity of processors, but does not address the heterogeneity and dynamicity of networks.

In this paper, a dynamic load balancing scheme for distributed systems is proposed. This scheme takes into consideration (i) the heterogeneity of processors and (ii) the heterogeneity and dynamic load of the networks. The DLB scheme addresses the heterogeneity of processors by generating a relative performance weight for each processor. When distributing workload among processors, the load is balanced proportional to these weights. To deal with the heterogeneity of network, our scheme divides the load balancing process into global load balancing phase and local load balancing phase. The primary objective is to minimize remote communication as well as to efficiently balance the load on the processors. In this paper, a heuristic method is proposed to evaluate the computational gain and the redistribution cost for global redistributions. The scheme addresses the dynamic features of networks by adaptively choosing an appropriate action based on the current observation of the traffic on the networks.

The remainder of this paper is organized as follows. Section 2 introduces Solution Adaptive Finite Element Graph Application and a parallel load balancing method Condensed Binary Tree Load Balancing (CBTLB) [6] method in a distributed environment. Section 3 describes our proposed dynamic load balancing scheme for distributed systems. Section 4 presents the experimental results comparing the performance by this distributed DLB scheme with CBTLB scheme which does not consider the heterogeneous and dynamic features of distributed systems. Finally, section 5 summarizes the paper.

2 Finite Element Graph Method

This section gives an overview of the FEG method, and CBTLB, a parallel load balancing method for FEG on distributed memory multicomputers. Additional details about CBTLB and FEG can be found in [6].

2.1 Layout of Finite Element Graph

In the finite element model, an object can be viewed as a finite element graph, which is a connected and undirected graph that consists of a number of finite elements. Each finite element is composed of a number of nodes. Due to the properties of computation-intensiveness and computation-locality, it is very attractive to implement the finite element method on distributed memory multicomputers [5, 14, 17, 19, 20]. When nodes of a solution-adaptive finite element graph were evenly distributed to processors by some mapping algorithms, according to the communication property of the finite element graph, we can get a processor graph from the partition. For example, Figure 1 shows a partition of a 21-node finite element graph on seven processors. The corresponding processor graph of Figure 1 is shown in Figure 2. In a processor graph, nodes represent the processors and edges represent the communication needed among processors. The weights associated with nodes and edges denote the computation and the communication costs, respectively.

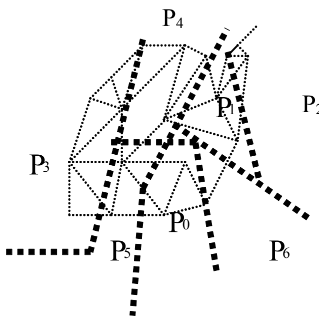

Fig. 1 A partition of 21 node finite element graph on 7 processors

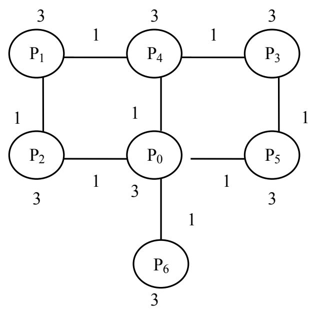

Fig. 2 The corresponding processor graph of Fig. 1

2.2 CBTLB: A Parallel Load Balancing Method

When a finite element graph is refined during run-time, it will result in load imbalance of processors. To balance the computational load of processors, the CBTLB method [6], work in the following five phases:

Phase 1: Obtain a processor graph *G* from the initial partition.

Phase 2: Group processors of *G* into metaprocessors to obtain a condensed processor graph G_c incrementally. Each metaprocessor of G_c is a hypercube. The metaprocessors in G_c are constructed as follows: First, a processor P_i with the smallest degree in *G* and a processor P_j that is a neighbor processor of P_i and has the smallest degree among those neighbor processors of P_i are grouped into a metaprocessor. Then, the same construction is applied to other ungrouped processors until there are no processors can be grouped into a hypercube. Repeat the grouping process to each metaprocessor until there are no metaprocessors can be grouped into a higher order hypercube.

Phase 3: Find a binary tree $T = (V, E)$ from G_c, where *V* and *E* denote the metaprocessors and edges of *T*, respectively. The method of constructing a binary tree is the same as that of the BTLB method.

Phase 4: Based on *T*, calculate the global load balancing information and schedule the load transfer sequence by using a similar TWA method for metaprocessors. Assume that there are *M* processors in a tree and *N* nodes in a refined finite element graph. We define *N/M* as the average weight of a processor. To obtain the global load balancing information, the quota and the load of each processor in a tree are calculated. The quota is defined as the sum of the average weights of processors in a metaprocessor C_i and processors in children processors of C_i. The load is defined as the sum of the weights of processors in a metaprocessor C_i and processors in children metaprocessors of C_i. The difference of the quota and the load of a metaprocessor is the number of nodes that a metaprocessor should send to or receive from its parent metaprocessor. If the difference is negative, a metaprocessor should send nodes to its parent metaprocessor. Otherwise, a metaprocessor should receive nodes from its parent metaprocessor. After calculating the global load balancing information, the schedule is determined as follows. Assume that *m* is the number of nodes that a metaprocessor C_i needs to send to another metaprocessor C_j. We have the following two cases:

Case 1: If the weight of C_i is less than *m*, the schedule of these two metaprocessors is postponed until the weight of C_i is greater than or equal to *m*.

Case 2: If the weight of C_i is greater than or equal to *m*, a schedule can be made between processors of C_i and C_j. Assume that *ADJ* denotes the set of processors in C_i that are adjacent to those in C_j. If the sum of the weights of processors in *ADJ* is less than *m*, a schedule is made to transfer nodes of processors in C_i to processors in *ADJ* such that the weights of processors in *ADJ* is greater than or equal to *m*. If the sum of the weights of processors in *ADJ* is greater than or equal to *m*, a schedule is made to send *m* nodes from processors in *ADJ* to those in C_j.

Phase 5: Perform load transfer (send/receive) among metaprocessors based on the global load balancing information, the schedule, and

T. The load transfer method is similar to that of the BTLB method. After performing load transfer process among metaprocessors, a dimension exchange method (DEM) [11] is performed to balance the computational load of processors in metaprocessors.

3. Distributed Dynamic Load Balancing Scheme

In this section, we present a DLB scheme for FEG applications on distributed systems. To address the heterogeneity of processors, each processor is assigned a relative weight. To deal with the heterogeneity of networks, the scheme divides the load balancing process into two steps: global load balancing phase and local load balancing phase. Further, the proposed scheme addresses dynamic feature of networks by adaptively choosing an appropriate action according to the traffic on them. The details are given in the following subsections.

3.1 Description

First, we define a “group” as a set of processors which have the same performance and share an intraconnected network; a group is a homogeneous system. A group can be a shared-memory parallel computer, a distributed-memory parallel computer, or a cluster of workstations. Communications within a group are referred as local communication, and those between different groups are remote communications. A distributed system is composed of two or more groups.

Our distributed DLB scheme entails two steps to redistribute the workload: global load balancing phase and local load balancing phase, which are described in detail below.

- Global Load Balancing Phase

For a solution-adaptive finite element application program, the number of nodes increases discretely due to the refinement of some finite elements during the execution. This may result in load imbalance of processors. So after each refinement, the scheme evaluates the load distribution among the groups by considering both heterogeneous and dynamic features of the system. If imbalance is detected, a heuristic method described in the following subsections is invoked to calculate the computational gain of removing the imbalance and the overhead of performing such load redistribution among groups. If the computational gain is larger than the redistribution overhead, this step will be invoked. All the processors will be involved in this process, and both global and local communications are considered. Workload will be redistributed by considering the heterogeneity of number of processors and processor performance of each group.

- Local Load Balancing

After each refinement, each group entails a balancing process within the group. The parallel DLB scheme as mentioned in section 2.2 is invoked, that is, the workload of each group is evenly and equally distributed among the processors. However, load balancing is only allowed within the group. An overloaded processor can migrate its workload to an underloaded processor of the same group only. During this step, load imbalance may be detected among groups; however, the global balancing process will not be invoked until the next refinement.

3.2 Cost Evaluation

To determine if a global redistribution is invoked, an efficient evaluation model is required to calculate the redistribution cost and the computational gain. The evaluation should be very fast to minimize the overhead imposed by the DLB. Basically, the redistribution cost consists of both communicational and computational overhead. The communicational overhead includes the time to migrate workload among processors. The computational overhead includes forming the groups of metaprocessors, calculating load balancing information and performing a load transfer algorithm to balance the computational load of metaprocessors.

We propose a heuristic method to evaluate the redistribution cost as follows. First, the scheme checks the load distribution of the system. If imbalance exists, the scheme calculates the amount of load needed to migrate between groups. In order to adaptively calculate communication cost, the network performance is modeled by the conventional model, that is $T_{comm} = \alpha + \beta \times L$. Here T_{comm} is the communication time, α is the communication latency, β is the communication transfer rate, and L is the data size in bytes. Then the scheme sends two messages between groups, and calculates the network performance parameters α and β. If the amount of workload need to be redistributed is W, the communication cost would be $\alpha + \beta \times W$.

This communication model is very simple so little overhead is introduced. To estimate the computational cost, the scheme uses history information that is, recording the computational overhead of the previous refinement. We denote this portion of cost as δ. Therefore, the total cost for redistribution is:

$$Cost = \alpha + \beta \times W + \delta \qquad (1)$$

3.3 Gain Evaluation

The scheme predicts the computational gain by the following heuristic method. Between two refinement of finite element graph, the scheme records several performance data, such as the amount of load each processor has, the number of iterations performed during each refinement, and the execution time for one refinement. For each group, the total workload is calculated for one refinement using this recorded data. Then the difference of total workload between groups is estimated. Lastly, the computational gain is estimated by using the difference of total workload and the recorded execution time of one refinement. The detailed formula is as follows:

$$W_{group}(t) = [\sum_{proc \in group} w_{proc}(t)] \times N_{iter}(t) \qquad (2)$$

$$Gain = T(t) \times \frac{\max(W_{group}(t)) - \min(W_{group}(t))}{Number\_Groups \times \max(W_{group}(y))} \qquad (3)$$

Here, Gain denotes the estimated computational gain for global load balancing at time t; $w_{proc}(t)$ is the workload of processor proc for time t; $W_{group}(t)$ is the total amount of load of group for time t; $N_{iter}(t)$ is the number of iterative steps for last refinement; and T (t) is the execution time for last refinement. Hence, the gain provides a very conservative estimate of the amount of decrease in execution time that will occur from the redistribution of load resulting from the DLB.

3.4 Global Load Redistribution

The global load redistribution is invoked when the computational gain is larger than some factor times the redistribution cost, that is, when Gain > γ × Cost. Here, γ is a user-defined parameter (default is 2:0) which identifies how much the computational gain must be for the redistribution to be invoked. The detailed sensitivity analysis of this parameter will be included in our future work. During the global redistribution step, the scheme redistributes the workload by considering the heterogeneity of processors. For example, suppose the total workload is W, which needs to be partitioned into two groups. Group A consists of n_A processors and each processor has the performance of p_A; group B consists of n_B processors and each processor have the performance of p_B. Then the *global balancing* process will partition the workload into two portions: $W \times \frac{n_A \times p_A}{n_a \times p_A + n_B \times p_B}$ for group A and

$W \times \frac{n_B \times p_B}{n_B \times p_B + n_A \times p_A}$ for group B.

Basically, this step entails moving the groups' boundaries slightly from underloaded groups to overloaded groups so as to balance the system.

4 Experimental Results

In this section we compare the performance of CBTLB executed on a distributed environment with homogeneous processors with that executed on a distributed environment with heterogeneous processors. To compare the performance of the load-balancing methods, the algorithms have been implemented with some simulation programs. The criteria used to evaluate the performance are execution time and the number of processes to be migrated to balance the system load.

CBTLB method has been implemented in distributed systems of homogeneous processors. The execution time and the number of processes to be migrated of CBTLB methods, with 7, 15, 25, 30, and 40 homogeneous processors are shown in Table 1.

The proposed DLB method has been implemented in distributed systems of heterogeneous processors. The execution time and the number of processes to be migrated of new CBTLB methods, with 7, 15, 25, 30, and 40 heterogeneous processors are shown in Table 2.

Table 1: The execution time in seconds and the number of processes to be migrated of CBTLD method for different load samples with different number of homogeneous processors.

No. of Processors	Execution time in seconds	No. of processes to be migrated
7	1.500549	782
15	1.500549	2286
25	1.600455	4426
30	1.600655	5556
40	1.612354	7355

Table 2: The execution time in seconds and the number of processes to be migrated of proposed DLB method for different load samples with different number of heterogeneous processors.

No. of Processors	Execution time in seconds	No. of processes to be migrated
7	1.600655	785
15	1.600665	2170
25	1.612365	4120
30	1.700236	5450
40	1.710010	7255

Figure 3 compares the total execution times with varying configurations for both load balancing schemes. It is observed that the total execution time and the total number of processes needed to be migrated by using the proposed DLB method is near same to some cases as compared to using CBTLB method. The most noteworthy reveal about the proposed DLB method is that it takes into account the heterogeneity of the processors of the distributed environment which CBTLB method cannot.

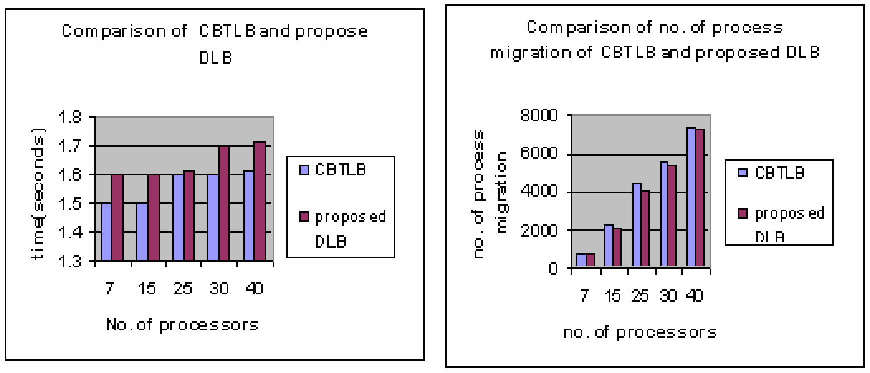

Figure 3: execution time and no. of process migration for CBTLB and proposed DLB method

5 Conclusion

In this paper, we proposed a dynamic load balancing scheme for distributed systems. This scheme takes into consideration (i) the heterogeneity of processors and (ii) the heterogeneity and dynamic load of networks. To address the heterogeneity of processors, each processor is assigned a relative performance weight. When distributing workload among processors, the load is distributed proportionally to these weights. To deal with the heterogeneity of network, the scheme divides the load balancing process into *global load balancing phase* and *local load balancing phase*. Further, the scheme addresses the dynamicity of networks by adaptively choosing an appropriate action based on the observation of the traffic on the networks. For global redistribution, a heuristic method was proposed to evaluate the computational gain and the redistribution cost. The experiments, however, illustrate the advantages of our DLB to handle the heterogeneity and dynamic load of the networks. The experiments show that by using this distributed DLB scheme, the total execution time can be reduced by 9%-46% and the average improvement is more than 26%, as compared to using parallel DLB scheme which does not consider the heterogeneous and dynamic features of distributed systems.

Our future work will focus on including more heterogeneous machines and larger real datasets into our experiments. Further, we will connect this proposed DLB scheme with tools to get more accurate evaluation of underlying networks. Lastly, a detailed sensitivity analysis of parameters used in this distributed DLB scheme will also be completed.

References

[1] C. Aykanat, F. Ozg†ner, S. Martin, and S.M. Doraivelu, "Parallelization of a Finite Element Application Program on a Hypercube Multiprocessor," *Hypercube Multiprocesso*r, pp. 662-673, 1987.

[2] S.T. Barnard and H.D. Simon, "Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems," *Concurrency: Practice and Experienc*e, vol. 6, no. 2, pp. 101- 117, Apr. 1994.

[3] J.R. Gilbert and E. Zmijewski, "A Parallel Graph Partitioning Algorithm for a Message-Passing Multiprocessor," *Int'l J. Parallel Programmin*g, vol. 16, no. 6, pp. 427-449, 1987.

[4] H.D. Simon, "Partitioning of Unstructured Problems for Parallel Processing," *Computing Systems in Eng*., vol. 2, nos. 2/3, pp. 135- 148, 1991.

[5] R.D. Williams, "Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations," *Concurrency: Practice and Experienc*e, vol. 3, no. 5, pp. 457-481, Oct. 1991.

[6] Y.C. Chung and C.J. Liao, "Tree-Based Parallel Load Balancing Methods for Solution-Adaptive Finite Element Graphs on Distributed Memory Multicomputer", IEEE Transaction on Parallel and Distributed Systems, Vol. 10, No. 4, pp. 360-370, (1999).

[7] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. In *IEEE Transactions on Parallel and Distributed computing*, 7:279–301, October 1989.

[8] L. Oliker and R. Biswas. PLUM:parallel load balancing for adaptive refined meshes. In *Journal of Parallel and Distributed Computing*, 47(2):109–124, 1997.

[9] K. Schloegel, G. Karypis, and V. Kumar. Multilevel diffusion schemes for repartitioning of adaptive meshes. In *Journal of Parallel and Distributed Computing*, 47(2):109–124, 1997.

[10] M. Willebeek-LeMair and A. Reeves. Strategies for dynamic load balancing on highly parallel computers. In *IEEE Transactions on Parallel and Distributed Systems*, 4(9):979–993, Sep. 1993.

[11] C.Z. Xu, and F.C.M. Lau, "The Generalized Dimension Exchange Method for Load Balancing in k-ary n-cubes and Variants," J. Parallel and Distributed Computing, vol. 24, no. 1, pp. 72-85, 1995.

[12] M.R. Garey and D.S. Johnson, *Computers and Intractability, A Guide to Theory of NP-Completenes*s. San Francisco: Freeman, 1979.

[13] J.R. Gilbert, G.L. Miller, and S.H. Teng, "Geometric Mesh Partitioning: Implementation and Experiments," *Proc. Ninth Int'l Parallel Processing Symp*., Santa Barbara, Calif., pp. 418-427, Apr. 1995.

[14] R.D. Williams, *DIME: Distributed Irregular Mesh Environmen*t. California Inst. of Technology, 1990.

[15] K. Dragon and J. Gustafson. A low-cost hypercube load balance algorithm. In *Proc. Fourth Conf. Hypercubes, Concurrent Comput. and Appl.*, pages 583–590, 1989.

[16] R. Elsasser, B. Monien, and R. Preis. Diffusive Load Balancing Schemes for Heterogeneous Networks. In *Proc. SPAA'2000*, Maine, 2000.

[17] A. Sohn and H. Simon. Jove: A dynamic load balancing framework for adaptive computations on an sp-2 distributed multiprocessor. In *NJIT CIS Technical Report*, New Jersey, 1994.

[18] C. Walshaw. Jostle:partitioning of unstructured meshes for massively parallel machines. In *Proc. Parallel CFD'94*, 1994.

[19] I.G. Angus, G.C. Fox, J.S. Kim, and D.W. Walker, *Solving Problems on Concurrent Processor*s, vol. 2. Englewood Cliffs, N.J.: Prentice Hall, 1990.

[20] C. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salman, and D.W. Walker, *Solving Problems on Concurrent Processor*s, vol. 1. Englewood Cliffs, N.J.: Prentice Hall, 1988.

Maheen Islam received her B.Sc. and M.Sc. degrees in Computer Science from Dhaka University in 1998 and 1999, respectively. She was lecturer in the Department of Computer Science and Engineering at BRAC University from 2003 to 2005. In January 2005, she joined the Computer Science and Engineering Department of East West University, where she is currently a senior lecturer. Her research interests include distributed systems, load balancing for distributed systems, networks and communication.

Upama Kabir obtained her MSc degree from the Department of Computer Science, University of Dhaka, Bangladesh in 1999 and MS degree from the Department of Computer Science and Engineering, University of New South Wales, Australia in 2005. She worked as a Lecturer (1999-2003), as Assistant Professor (2003-2007) in the Department of Computer Science and Engineering, University of Dhaka, Bangladesh, where she is currently working as Associate Professor. She received two awards (Ashalata Sen Gold Medal and Nurunnessa Khatun Biddyabinodini Gold Medal) from Dhaka University for her MSc result. Her research interests include artificial intelligence, computer network, parallel and distributed systems. She has a number of research publications in national and international proceedings and journals.